AI Documentation Generator: Project Documentation

This document details the functionality and workflow of the "AI Documentation Generator" project, a Streamlit application leveraging Google's Generative AI to automatically generate documentation for code.

1. Overview

The application allows users to upload a ZIP file containing source code or directly paste code into a text area. It then uses the Google Gemini API to generate documentation, including an analysis of the code's workflow and relationships between modules, function descriptions, parameter details, and examples. The generated documentation is displayed in the Streamlit app and can be downloaded as a text file. The application handles large codebases by splitting them into smaller chunks before sending them to the API.

2. Modules and Workflow

The application consists of several interconnected modules:

- **Streamlit Interface (streamlit):** Provides the user interface for uploading files, pasting code, triggering the documentation generation process, and displaying the results.
- **Code Handling (zipfile, io):** Handles the uploading and processing of code from ZIP files or the text area. Extracts code from supported file types (.py, .cs, .java, .cpp, .js, .edmx) within the ZIP archive.
- **Google Generative AI Interaction (google.generativeai):** The core module responsible for interacting with the Google Gemini API to generate documentation. It sends code chunks to the API and retrieves the generated documentation. Includes error handling for API connection issues.
- **Text Processing (textwrap):** Formats the generated documentation for better readability.
- **Chunking Mechanism (split_code_into_chunks):** Divides large codebases into smaller, manageable chunks to avoid exceeding the API's token limit.

Workflow:

1. **User Input:** The user either uploads a ZIP file containing source code or pastes code directly into the text area.
2. **Code Extraction & Processing:** The application extracts code from the ZIP file (if uploaded) or uses the pasted code. It then splits the code into chunks using split_code_into_chunks.

3. **API Interaction:** Each code chunk is sent to the Google Gemini API via generate_documentation. The API generates documentation for each chunk. Error handling is implemented to manage potential connection issues.
4. **Documentation Assembly:** The generated documentation for each chunk is concatenated.
5. **Output:** The complete documentation is displayed in the Streamlit app and is available for download.

3. Function Descriptions

3.1 to_markdown(text)

Description: Converts plain text into a Markdown format by indenting each line with > . Replaces bullet points (•) with Markdown-style bullet points (*).

Parameters:

- text (str): The input text to convert.

Return Value:

- (str): The Markdown formatted text.

3.2 generate_documentation(code)

Description: Sends the provided code to the Google Gemini API to generate documentation. Handles potential API connection errors with retries.

Parameters:

- code (str): The code snippet to be processed.

Return Value:

- (str): The generated documentation from the API, or None if the API call fails after multiple attempts.

3.3 split_code_into_chunks(code, max_tokens=10000)

Description: Splits a large code string into smaller chunks, each containing approximately max_tokens words. This is crucial for handling large codebases that might exceed the Google Gemini API's token limits.

Parameters:

- code (str): The code to be split into chunks.
- max_tokens (int, optional): The approximate maximum number of words per chunk. Defaults to 10000.

Return Value:

- (list of str): A list of code chunks.

4. Streamlit App Structure

The Streamlit app uses the following components:

- st.title: Sets the title of the application.
- st.markdown: Displays markdown formatted text for instructions and output.
- st.file_uploader: Allows users to upload ZIP files.
- st.text_area: Provides a text area for users to paste code.
- st.button: Triggers the documentation generation process.
- st.error: Displays error messages.
- st.download_button: Provides a button to download the generated documentation.

5. Error Handling

The application includes error handling for:

- **API Connection Errors:** The generate_documentation function retries the API call multiple times before giving up.
- **Missing Code Input:** The application checks for code input and displays an error message if no code is provided.

6. Future Enhancements

- **Support for more file types:** Expand support beyond the currently supported programming languages.
- **Improved error reporting:** Provide more detailed error messages to the user.
- **Advanced formatting options:** Allow users to customize the output format of the documentation.
- **Integration with other AI models:** Explore integration with different large language models.
- **Code syntax highlighting:** Enhance readability by adding syntax highlighting to the displayed code and documentation.